PromptIntern: Saving Inference Costs by Internalizing Recurrent Prompt during Large Language Model Fine-tuning

Jiaru Zou[1†], Mengyu Zhou[2‡], Tao Li[3†], Shi Han[2], Dongmei Zhang[2]
[1] University of Illinois Urbana-Champaign [2] Microsoft
[3] Shanghai Jiao Tong University
jiaruz2@illinois.edu, li.tao@sjtu.edu.cn,
{mezho, shihan, dongmeiz}@microsoft.com

Abstract

Recent advances in fine-tuning large language models (LLMs) have greatly enhanced their usage in domain-specific tasks. Despite the success, fine-tuning continues to rely on repeated and lengthy prompts, which escalate computational expenses, require more resources, and lead to slower inference. In this paper, we present a novel approach, PromptIntern, which internalizes prompt knowledge during model fine-tuning to achieve efficient inference and save costs. Instead of compressing the prompts for a vanilla model, PromptIntern aims to embed the recurrent prompt directly into the model parameters. We design a fine-tuning pipeline that includes instruction template compression, few-shot example absorption, and a progressive internalization strategy, effectively diminishing the need for intricate prompts during inference. Comprehensive experiments on challenging NL2Code tasks demonstrate that our method reduces input tokens by more than 90%, accelerates inference by 4.2 times, and reduces monetary inference costs by 88.3%.

1 Introduction

Large language models (LLMs) have become pivotal in numerous natural language processing (NLP) applications, such as natural language generation (Dong et al., 2019; Zheng et al., 2024), reasoning (Zhu et al., 2023; Sui et al., 2023), and code generation (Luo et al., 2023b; He et al., 2024; Rozière et al., 2024). To enhance the predictive accuracy of LLMs in domain-specific tasks, recent techniques in fine-tuning, such as parameter-efficient fine-tuning (PEFT) (He et al., 2021; Hu et al., 2021; Lester et al., 2021a), have been developed for pretrained models to excel in specific tasks by adjusting their parameters to better align with targeted datasets (Hu et al., 2021). Many of these fine-tuning approaches typically adopt prompts that are optimized and integrated with detailed instructions, examples, and retrieved documents through prompt engineering techniques such as chain-of-thought (Wei et al., 2022), few-shot prompting (Brown et al., 2020), and retrieval-augmented generation (Lewis et al., 2020; Cheng et al., 2023).

[†] The contributions by Jiaru Zou and Tao Li have been conducted and completed during their internships at Microsoft.
[‡] Corresponding author.

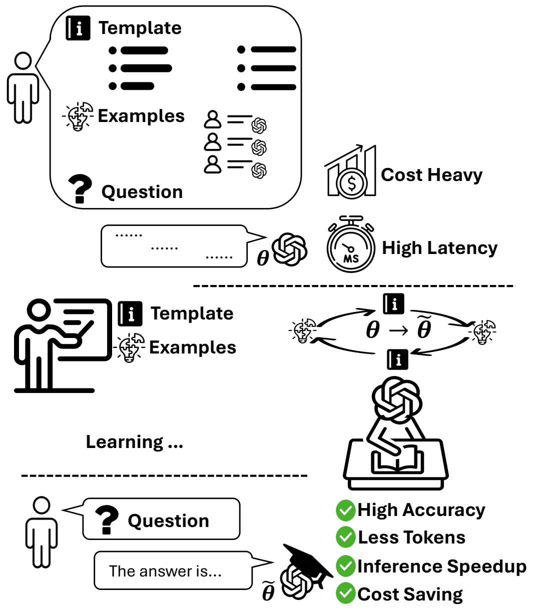


Figure 1: An illustration of PromptIntern: Like human interns, LLMs learn and internalize repeated prompt information such as templates and examples during fine-tuning, leading to efficient and effective inference.

Although these advancements enhance the capabilities of LLMs during fine-tuning, they also present new challenges: Prompt engineering often necessitates longer prompts, and directly integrating lengthy prompts into the training process further increases computational costs during inference (VM et al., 2024). This increase in cost precludes LLMs in many cost-sensitive scenarios where computational resources are constrained. Several prompt compression methods (Li et al., 2023; Jiang et al., 2023a; Pan et al., 2024) have been proposed to reduce text redundancy. They design various prompt compression systems and strive to preserve maximum information between original and compressed prompts. While these methods ensure the retention of original prompt information, they mainly focus on the prompt perplexity and overlook the adaption of target LLMs during compression (Pan et al., 2024). For challenging tasks that require model fine-tuning (Mosbach et al., 2023), these approaches struggle to establish connections between compressed tokens and dynamically adjusted model parameters. Consequently, naively applying these compression methods often leads to large performance degradation, as relevant information may be inadvertently removed or distorted during the compression process.

In this paper, we propose a novel prompt internalization approach, namely **PromptIntern**, which internalizes prompt input during model fine-tuning and enables efficient inference. Unlike prompt compression which removes tokens based solely on prompt information entropy, we aim to transfer various types of prompt knowledge into updated model parameters, thereby directly enhancing LLMs' understanding. Our idea is motivated by the human learning process as illustrated in Figure 1: During internship on-boarding, human interns need detailed instructions, examples, and documents to learn new tasks. As they internalize these materials and become familiar with their duties, they master the necessary skills and no longer require extra guidance. Similarly, when specific prompt information (e.g. task constraints, output formats/styles) is repeatedly exposed to an LLM during fine-tuning, the model can gradually internalize the knowledge into its updated parameters. Such repeated information can be progressively eliminated from prompt inputs since it becomes unnecessary for inference of the master LLM.

We dub our approach **PromptIntern** to regard LLMs as human **intern**s and **intern**alize prompt knowledge progressively. Our approach consists of several key steps: Initially, we classify an input prompt into three components: the template, examples, and query. We start by setting a schedule to linearly decrease both the template compression rate and the number of few-shot examples across training stages. Following the schedule, we implement template compression and example absorption to pre-process the input prompts. We then introduce a comprehensive pipeline that enables LLMs to progressively internalize template and example components into model parameters during fine-tuning and efficiently perform inference using query-only prompts.

We assess our method on challenging NL2Code tasks (Zan et al., 2022) that are widely recognized as benchmarks for model fine-tuning. Our experiments evaluate PromptIntern on three key metrics: accuracy, token usage, and inference speed. The results indicate that under identical fine-tuning settings, our method not only surpasses prompt compression methods but also achieves comparable accuracy to direct fine-tuning. Moreover, it accelerates the inference process by a factor of 4.2 and reduces token usage by over 90% compared to direct fine-tuning. These enhancements demonstrate that our approach successfully balances efficiency and effectiveness, making it well-suited for optimizing LLM performance across various cost-saving scenarios. We further quantify the total monetary cost savings and conduct detailed analyses to elucidate the efficacy of our approach and provide insights into its underlying mechanisms. Our main contributions can be summarized as follows:

- We proposed *PromptIntern*[1], a novel prompt internalization method that aims to internalize repetitive prompt knowledge into the model's parameters, achieving high inference efficiency while maintaining model performance.
- We devised detailed prompt internalization strategies for template compression and example absorption along with a tailored progressive fine-tuning pipeline.
- We conducted extensive experiments with detailed analyses on challenging NL2Code tasks. The experimental results show that our approach reduces token usage by over 90%, speeds up inference time by 4.2 times, and achieves 88.3% cost savings across a broad spectrum of LLMs.

2 Related Work

Prompt compression rephrases original prompts more concisely and is classified into task-aware and task-agnostic approaches. Specifically, the task-aware approaches, like LongLLMLingua (Jiang et al., 2023b), utilize a question-aware coarse-to-fine-grained strategy to compress information based on the query. In contrast, methods like soft prompts (Wingate et al., 2022;

[1]Our code will be released at https://github.com/microsoft/PromptIntern

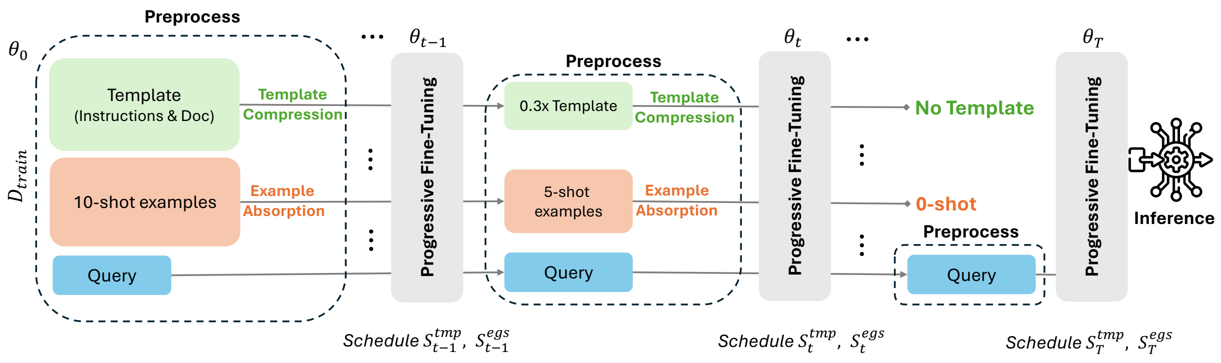


Figure 2: Overview of PromptIntern framework. We structure the input prompt into three components: the template, examples, and query. By employing template compression and example absorption, we efficiently preprocess each component based on schedule $\mathcal{S}^{tmp}, \mathcal{S}^{egs}$. We then use a progressive fine-tuning strategy to gradually incorporate prompt knowledge into the model parameters θ, facilitating efficient inference without sacrificing performance.

Liu et al., 2022; Mu et al., 2024) use learnable tokens to condense prompts. Conversely, the task-agnostic methods utilize metrics such as information entropy to eliminate redundant prompt information, with systems like LLMLingua (Jiang et al., 2023a; Li et al., 2023) estimating token importance using a smaller model. Despite the demonstrated effectiveness of these methods, producing compressed text that can generalize across different tasks and be effectively integrated into training scenarios remains a challenge.

Model fine-tuning adopts pre-trained LLMs to specific tasks by modifying parameters. Based on the assumption that fine-tuning adds less new information to the model pre-trained on large internet-scale datasets, Parameter-Efficient Fine-Tuning (PEFT) methods aim to curtail the costs of tuning large models by adjusting a subset of parameters. Existing PEFT methods can be broadly categorized into three main approaches: 1) Adapter-based methods (Houlsby et al., 2019; He et al., 2021): Introduce trainable modules within a static "backbone" network, offering flexibility but potentially increasing model size. 2) Prompt-based methods (Lester et al., 2021b; Razdaibiedina et al., 2023; Nashid et al., 2023): Employ trainable "soft tokens" at input sequence start, requiring effective prompt design per task. 3) Low-rank adaptation methods (Hu et al., 2021; Dettmers et al., 2024; Liu et al., 2024): Use low-rank matrices to approximate required weight adjustments, avoiding additional inference burden and often delivering strong performance. Despite advancements in fine-tuning strategies, data inputs should be carefully managed and distinguished from lengthy ones used for direct inference.

3 Problem Formulation

Let an input prompt as $x = (x^{tmp}, x^{egs}, x^{que})$, where each input prompt x is considered as a tuple of three components: x^{tmp} as the template such as fixed instructions, API docs, etc., x^{egs} as the examples, and x^{que} as the query. Typically, x^{tmp} and x^{egs} are relatively fixed and lengthy but essential for complex tasks. Let $f_\theta(\cdot)$ denote the neural network function of a LLM model, typically transformer (Vaswani et al., 2017), parameterized by θ. The generated output by LLM can be represented as $f_\theta(x)$.

We then consider the following problem of prompt internalization. Given a training dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^n$ where n is the number of training samples, x_i is an input prompt defined above, and y_i is the corresponding groundtruth output. Our goal is to internalize the knowledge contained in templates and examples of each input prompt i.e. $\{(x_i^{tmp}, x_i^{egs})\}_{i=1}^n$ into model parameters θ during fine-tuning, enabling efficient inference while maintaining high prediction performance through $\{x_i^{que}\}_{i=1}^n$ only. Formally, the prompt internalization objective can be formulated as follows:

$$\min_{\tilde{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{\tilde{\theta}}(x_i^{que})\right) \tag{1}$$

where $\mathcal{L}(\cdot)$ denotes the loss function and $\tilde{\theta}$ denotes the updated weights with internalized prompt knowledge. For a new incoming prompt only containing the query, the updated LLM with $f_{\tilde{\theta}}(\cdot)$ can internally recover the output without the assistance of instruction and examples.

4 Methodology

In this section, we introduce our method PromptIntern in detail. We first present the template compression to compress the entire fixed template part inside a prompt. Then we show the example absorption to effectively absorb demonstration examples into model parameters. Finally, we introduce a tailored training strategy for PromptIntern. The overall framework is shown in Figure 2.

4.1 Template Compression

We first introduce template compression, which is designed to compress the common template information exists across training instances. The motivation of the template compression stems from the following aspects: 1) Redundancy. The instruction is repetitive across prompts for a given task, often containing unnecessary tokens that do not contribute to the language model's understanding, posing significant memory and computational burdens when the instruction is lengthy; and 2) Noise. Excessively long prompts may incorporate extraneous elements—either irrelevant or misleading information—that serve as noise and can adversely affect the model's generation.

To mitigate the issues stated above, we propose a template compression system, which can generally be expressed as:

$$\tilde{x}^{\text{tmp}} = C(x^{\text{tmp}}, \tau^{\text{tmp}}) \tag{2}$$

where C is a specific template compressor, $\tilde{x}^{tmp}$ is the compressed template, and τ^{tmp} is the template compression rate as defined in (Jiang et al., 2023a), varying at differnt training interations. We then adopt a predetermined schedule $\mathcal{S}^{tmp}(t)$ to progressively reduce and internalize the prompt template information during the t-th training iteration. Specifically, for a total of T training iterations, we initially set τ^{tmp} to 1 at $\mathcal{S}^{tmp}(0)$ and gradually decrease the value of τ^{tmp} at $\mathcal{S}^{tmp}(t)$ to zero at end to achieve fully template internalization. Note that such a compression system is also flexible, allowing it to halt at a desired non-zero compression rate. This flexibility allows to maintenance of a certain level of compressed template, serving as a trade-off to preserve inference accuracy in specific scenarios, as discussed in Section 5.4. In addition to the progressively decreasing template schedule, we also specify the template compressor C for better utilization. we categorize it into two types which exactly reflect the primary components of the template defined in the problem formulation: the instruction compressor and document compressor:

Instruction Compressor targets the static elements within prompts, specifically focusing on the instructional content. Instructions in training data often consist of repeated directives, guidelines, or predefined tasks which are common across multiple training scenarios. The primary goal of the instruction compressor is to distill these instructions down to their essential components, eliminating verbosity and redundancy without compromising the clarity or intent of the instructions.

Document Compressor is designed to handle the bulkier and more detailed portions of the prompts, such as API documentation or static demonstrations. These sections typically include extensive technical descriptions and examples that, while informative, often contain a significant amount of repetitive or non-essential information (Xu et al., 2023). The goal of the document compressor is to reduce the information unnecessary for understanding and applying the technical content, thereby streamlining the training process.

4.2 Example Absorption

Incorporating few-shot examples into fine-tuning not only improves information retrieval and memory recall (Hübotter et al., 2024) but also yields substantial benefits in handling a variety of tasks with minimal data input (Mosbach et al., 2023; Snell et al., 2017). However, directly adding lengthy few-shot examples to input prompts burdens the context window and increases inference latency. Motivated by this, we propose example absorption to benefit from the enhanced performance afforded by few-shot examples while preventing incurring significant additional overhead. Specifically, the example absorption mainly contains two stages: example retrieval and example removal.

Example Retrieval is designed to identify and select the most related few-shot examples from the training dataset and incorporate them into each training instance. The underlying rationale is to choose examples that closely align with the training instance so as to accelerate model's internalization during training. We employ a straightforward approach that utilizes a relevance scoring function $s(\cdot, \cdot)$ to assess the similarity between examples and the training instance. Specifically, we select the top k examples, varying at different training iterations, with the highest relevance scores to serve as our few-shot examples. For a training instance (x_i, y_i) with x_i being the input prompt and y_i being the corresponding groundtruth output, the selection process can be expressed as follows:

$$x_i^{egs} = \{(x_j, y_j) \mid j \neq i, s(y_i, y_j) \in \text{top } k \text{ scores}\} \tag{3}$$

Note that the scoring function is calculated based on common similarity metrics (Rubin et al., 2022; Chen et al., 2022; Dai et al., 2022). In our experiment, we use the BLEU as the scoring function.

Example Removal aims to progressively internalize the prompt knowledge from few-shot examples into model parameters. To achieve this, we also adopt a predetermined schedule $\mathcal{S}^{egs}(t)$ to gradually decrease the number of demonstration examples in each prompt instance during the t-th iteration. Specifically, for a total of T training iterations, we initially set k examples at $\mathcal{S}^{egs}(0)$ and then gradually decrease the value of k at each $\mathcal{S}^{egs}(t)$ to zero at end in order to achieve fully example internalization.

4.3 PromptIntern Pipeline

In this subsection, we describe the detailed pipeline of PromptIntern. As demonstrated in Algorithm 1, PromptIntern consists of three stages: preprocess (line 1-7), progressive fine-tuning (line 8-12), and inference (line 13-14).

Preprocess. For the first step, We preprocess the input prompts to prepare them for the progressive training stage. Specifically, we process the prompt template to different compression rates based on the schedule $\mathcal{S}^{tmp}(t)$ and retrieve examples for each training instance based on the schedule $\mathcal{S}^{egs}(t)$. For better illustration, we provide an example of a pre-processed prompt with respect to schedule in Appendix C.

Progressive Fine-tuning. We then fine-tune the model parameters for internalizing. Given the training iteration t, we update the model parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{b}\sum_{i=1}^{b} \nabla_\theta \mathcal{L}\big(f_{\theta_t}(x_i^{tmp}(t), x_i^{egs}(t), x_i^{que}), y_i\big) \tag{4}$$

where η is the learning rate, $\mathcal{L}$ is the cross-entropy loss function, b is the batch size, $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{b}$ is the data batch, and y is the groundtruth label.

Inference. After the progressive fine-tuning, we have trained the LLMs with updated model parameters θ_T to perform inference without adding instructions or any examples. Thus, we can predict the output simply with $f_{\theta_T}(x^{que})$.

Algorithm 1 PromptIntern Pipeline

```
Input: A training dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^{n}$
    with $x_i = (x_i^{tmp}, x_i^{egs}, x_i^{que})$ and corresponding labels $y_i$, A language model $f$ with initial
    parameters $\theta$, learning rate $\eta$, training iterations $T$, template compression schedule $\mathcal{S}^{tmp}$,
    example absorption schedule $\mathcal{S}^{egs}$
Output: The inference output $f_{\theta_T}(x^{que})$
 1: Preprocess
 2: for $i = 1, 2, \dots, n$ do
 3:     Obtain each $\tau^{tmp}$ from $\mathcal{S}^{tmp}$
 4:     Obtain each $k$ from $\mathcal{S}^{egs}$
 5:     Compress $x_i^{tmp}$ w/ each $\tau^{tmp}$ via Eq. (2)
 6:     Retrieve $k$ examples $x_i^{egs}$ via Eq. (3)
 7: end for
 8: Progressive Finetuning
 9: for $t = 0, 1, \dots, T-1$ do
10:     Adjust prompts with $\mathcal{S}^{tmp}(t)$ and $\mathcal{S}^{egs}(t)$
11:     Update model parameters $\theta_t$ via Eq. (4)
12: end for
13: Inference
14: Perform inference with $f_{\theta_T}(x^{que})$
```

Our objective is to effectively compress and incorporate prompt knowledge into model parameters that are specifically tailored for distinct tasks. In pursuit of this goal, we have adopted PEFT during the fine-tuning phase of PromptIntern. Specifically, we apply LoRA (Hu et al., 2021) as it imposes no additional computational costs during inference and allows for scalable deployment across multiple tasks (Sheng et al., 2023). Note that our outlined pipeline in Algorithm 1 is also compatible with other PEFT techniques.

5 Experiment

In this section, we evaluate the performance of PromptIntern across various benchmarks on the NL2Code task. The NL2Code task is widely recognized for its utility in evaluating LLMs on both fine-tuning efficacy and cost-effectiveness in real-world applications (Zan et al., 2022). Following this, our experiments primarily focus on two key perspectives: **1) Effectiveness**: assessing the performance accuracy of PromptIntern during inference phases; **2) Efficiency**: quantifying the reduction in token usage and corresponding cost savings achievable through PromptIntern.

Table 1: Comparison with prompt compression baselines on NL2Code benchmark. To ensure a fair comparison, we apply each baseline to compress the input prompt and use the compressed prompt as both training and testing data during model fine-tuning. We also standardize the compression ratio across all methods to approximately the same as indicated by $1/\tau_{all}$.

Methods	MBPP			NL2F			NL2Bash		
(*Inference on GPT-3.5*)	Pass@1	Input Tokens	$1/\tau_{all}$	E.M.	Input Tokens	$1/\tau_{all}$	BLEU	Input Tokens	$1/\tau_{all}$
GPT4 Generation	61.8	128	1.8x	59.6	425	1.6x	59.5	256	1.9x
Selective Context	59.7	102	2.2x	56.4	391	1.7x	55.2	158	3.1x
LLMLingua	70.3	115	2.0x	64.2	417	1.6x	61.3	154	3.1x
LongLLMLingua	65.2	121	1.9x	67.8	425	1.6x	58.4	133	3.6x
LLMLingua-2	72.5	107	2.1x	70.4	407	1.7x	62.8	141	3.4x
PromptIntern	**78.1**	107	2.1x	**81.4**	407	1.7x	**70.5**	141	3.4x

5.1 Settings

Datasets We apply three typical NL2Code datasets: MBPP (Austin et al., 2021) for NL to python code generalization, NL2F (Zhao et al., 2024) for NL to Excel spreadsheet formulas generation, NL2Bash (Lin et al., 2018) for NL to Bash Shell commands generation. Please refer to Appendix A.1 for the dataset details.

Evaluation Metrics We use one-shot pass accuracy $Pass@1$ (Austin et al., 2021) for MBPP, *Exact Match (E.M.)* for NL2F, and *BLEU* score for NL2Bash. We also calculate the input tokens usage and compression ratio $1/\tau$ for each dataset.

Baselines We consider two types of baselines with setups below:
1) *Prompt Compression approaches.* We employ the latest advancements in prompt compression techniques. Specifically, we utilize Gist Tokens (Mu et al., 2024), GPT-4 Generation (Jiang et al., 2023b), Selective Context(Li et al., 2023), and LLMLingua series (Jiang et al., 2023a,b; Pan et al., 2024). Each prompt compression method is initially applied to compress the entire dataset to a predetermined compression rate. Then, the compressed dataset is utilized for both fine-tuning and inference evaluation.
2) *Direct Fine-tuning approaches.* We use "Direct" as the counterpart to our progressive fine-tuning strategy. Specifically, we adopt several conventional direct fine-tuning configurations, including i) direct fine-tuning with complete template and examples (e.g. *Template with 5-shots* in Table 2), ii) direct fine-tuning with compressed template and reduced examples (e.g. *Template x0.6 with 2-shots* in Table 2), iii) direct fine-tuning with template only (*Template only*), and iv) direct fine-tuning without template and examples (*No template*).

Models To demonstrate the broad applicability of PromptIntern, we utilize both closed-source and open-source LLMs with different parameter sizes for fine-tuning and inference processes.1) Closed-Source: We apply GPT-4-0613 (OpenAI, 2023), abbreviated as GPT-4, and GPT-3.5-turbo-0125[2], abbreviated as GPT-3.5. 2) Open-Source: We apply Mixtral-8x7B-v0.1 (Jiang et al., 2024), abbreviated as Mixtral-8x7B, Llama2-7B (Touvron et al., 2023), and Llama2-13B (Touvron et al., 2023).

Implementation Details Please refer to Appendix A for the additional experiments settings and implementation details.

5.2 Prompt Compression Comparison

Table 1 reports the overall result of PromptIntern with the prompt compression baselines inferenced on GPT-3.5 across all datasets. Here we establish the template compression rate τ_{tmp} at 0.3 across all prompt compression approaches as well as PromptIntern to ensure a fair comparison. And τ_{all} in the table represents the overall prompt's compression rate. We observe that while utilizing a comparable number of tokens for inference, our approach outperforms all baselines, achieving improvements of 5.6% on MBPP, 11.0% on NL2F, and 7.7% on NL2Bash. The result demonstrates that PromptIntern generally offers the best balance of efficiency and effectiveness across varied tasks. Note that since the Gist Token(Mu et al., 2024) baseline is only applicable to open-source LLMs, we separately compare it with our approach which can be found in Appendix A.3.

[2]https://platform.openai.com/docs/models/gpt-3-5-turbo

Table 2: Comparison with direct fine-tuning baselines on NL2Code benchmark.

MBPP										
Model	Template *with* 5-shots		Template x0.6 *with* 2-shots		Template Only		No Template		PromptIntern	
	Pass@1	Input Tokens	Pass@1	Input Tokens	Pass@1	Input Tokens	Pass@1	Input Tokens	Pass@1	Input Tokens
GPT-4	91.6	1181	87.4	424	87.3	226	77.2	43	87.9	43
GPT-3.5	82.7	1181	76.2	424	75.3	226	65.8	43	76.6	43
Mixtral-8x7B	69.8	1263	65.8	453	65.7	238	56.3	54	66.3	54
Llama2-13B	39.2	1286	37.5	471	36.4	251	26.4	58	37.1	58
Llama2-7B	30.4	1286	27.7	471	27.3	251	18.3	58	27.9	58
NL2F										
Model	Template *with* 10-shots		Template x0.6 *with* 5-shots		Template Only		No Template		PromptIntern	
	E.M.	Input Tokens	E.M.	Input Tokens	E.M.	Input Tokens	E.M.	Input Tokens	E.M.	Input Tokens
GPT-4	94.8	3540	92.1	1838	89.7	680	82.5	286	91.6	286
GPT-3.5	85.5	3540	78.1	1838	76.2	680	70.4	286	78.4	286
Mixtral-8x7B	69.3	4204	66.3	2191	63.8	814	54.2	339	65.2	339
Llama2-13B	59.2	4202	54.9	2183	54.1	812	32.9	339	55.3	339
Llama2-7B	45.4	4202	40.7	2183	38.5	812	21.8	339	40.8	339
NL2Bash										
Model	Template *with* 10-shots		Template x0.6 *with* 5-shots		Template Only		No Template		PromptIntern	
	BLEU	Input Tokens	BLEU	Input Tokens	BLEU	Input Tokens	BLEU	Input Tokens	BLEU	Input Tokens
GPT-4	86.7	1063	81.3	810	78.6	484	71.2	52	82.5	52
GPT-3.5	74.2	1063	67.5	810	65.1	484	61.2	52	67.7	52
Mixtral-8x7B	63.8	1320	58.3	1053	54.9	603	47.6	68	57.2	68
Llama2-13B	47.1	1244	43.9	988	41.6	574	35.1	64	43.5	64
Llama2-7B	35.8	1244	32.7	988	31.4	574	22.1	64	31.6	64

5.3 Direct Fine-tuning Comparison

Table 2 shows the comparison of our approach with direct fine-tuning baselines on three datasets. In MBPP, our method outperforms the *No Template* and *Template* baselines by 9.6%-10.8% and 0.6%-1.3%, respectively, and achieves similar results to *Template x0.6 with 2-shots*, using fewer tokens. We also reduce input token usage by 9.8x-12.2x, compared to the *Template with 5-shots*, which requires 22.2x-27.4x more tokens. In NL2F, our method improves over *No Template* by 8.0%-19.0% with the same token usage, matching the *Template x0.6 with 2-shots* baseline while reducing token usage by 6.4x. We also observe that larger models (LLama2-13B) show less degradation without prompt templates compared to smaller models (LLama2-7B), with a difference of -5.7% verses -21.2%. In NL2Bash, our approach exceeds *No Template* by 7.3%-11.3% and reduces token usage by 15.5x compared to the baseline method *Template x0.6 with 5-shots*.

5.4 Ablation Study

To effectively assess the impact of various components within *PromptIntern*, we introduce three variants of PromptIntern for ablation studies:

- **PromptIntern w/τ_{tmp}=0.3**, where we set the compression rate to 0.3 instead of 0 in template compression.
- **PromptIntern w/o Example Absorption**, where we omit the example absorption for retrieving and internalizing few-shot examples during fine-tuning.
- **PromptIntern w/o Template Compression**, where template compression is excluded for both fine-tuning and inference prompt instances.

The overall results is shown in Table 3. When comparing PromptIntern with PromptIntern w/$\tau_{tmp} = 0.3$, we observe an average of 2.4% drop on performance but a 3.7x compression ratio on token usages across all three datasets. This highlights the balance between compression rate and accuracy performance. When comparing *our* with *our* w/o Example Absorption, we observe a large performance drop in the latter variant, despite both approaches utilizing the same number of tokens for inference. The result demonstrates the importance of example absorption in internalizing essential information during the fine-tuning progress. In addition, when comparing PromptIntern with PromptIntern w/o Template Compression, we note that adding the template compression saves an average of 280 tokens across the datasets but experiences an average performance drop of 5%. The result above demonstrates that while internalizing the template into model parameters reduces token

Table 3: Ablation Study of PromptIntern.

Methods	MBPP			NL2F			NL2Bash		
(*Inference on GPT-3.5*)	Pass@1	Tokens	$1/\tau_{all}$	E.M.	Tokens	$1/\tau_{all}$	BLEU	Tokens	$1/\tau_{all}$
PromptIntern	76.6	**43**	5.3x	78.4	**286**	2.4x	67.7	**52**	9.3x
w/ τ_{tmp} = 0.3	78.1	107	2.1x	81.4	407	1.7x	70.5	241	2.0x
w/o Example Absorption	72.9	**43**	5.3x	73.5	**286**	2.4x	64.6	**52**	9.3x
w/o Template Compression	**80.2**	226	1.0x	**83.6**	680	1.0x	**73.5**	484	1.0x

Table 4: Comparison of schedule pattern and example retrival bank of PromptIntern. The results are inferenced on GPT-3.5.

PromptIntern	MBPP(Pass@1)	NL2F(E.M.)	NL2Bash(BLEU)
Pattern of Schedule $\mathcal{S}$			
- exp	74.8	72.5	59.4
- exp^{-1}	67.3	64.9	52.8
- linear (**ours**)	**77.6**	**78.4**	**67.7**
Example Retrival Bank			
- 25%	75.9	77.5	66.2
- 50%	76.1	78.1	66.8
- 100% (**ours**)	**77.6**	**78.4**	**67.7**

usage, it requires a trade-off in terms of inference performance.

5.5 Analysis on Schedule Pattern

In Table 4, we test the effectiveness of different scheduling patterns during the progressive fine-tuning process, specifically focusing on how the decreasing speed curve influences the compression of the template and absorption of few-shot examples. The patterns tested include exponential, inverse-exponential, and linear decrease.

From the data in the table, we observe that the linear decreasing schedule delivers the most consistent and highest performance across all three evaluation metrics, indicating superior performance in both parsing efficiency and language model understanding. Conversely, the inverse-exponential schedule shows the least effectiveness, with scores considerably lower in all metrics compared to the linear schedule. The exponential decrease performs moderately, but still falls short of the linear schedule, suggesting that a steady, predictable reduction is more beneficial than more aggressive decrease. This analysis suggests that for adopting a linearly decreasing schedule for progressive fine-tuning may lead to better performance in terms of accuracy compared to other scheduling patterns.

5.6 Analysis on Examples Retrieval Bank

Table 4 examines the impact of varying proportion of the training set used for constructing relevant examples in the examples retrieval bank. The options tested include using 25%, 50%, and 100% of the training set. The results show a trend where increasing the percentage of the training set used in the examples retrieval bank correlates with improved performance. This suggests that larger examples retrieval bank provides a richer set of few-shots for the model to learn from, thereby enhancing its ability to perform accurately across tasks.

Additional Experiments. In Appendix A.4, we explain why performing direct inference with a compressed prompt, while using the full prompt during training, is less effective compared to the progressive fine-tuning strategy employed by PromptIntern. Also, to provide a comprehensive analysis of PromptIntern's efficiency, we evaluate the inference speed and calculate the overall monetary cost during inference (details in Appendix A.5). In addition, to show that the principle of PromptIntern can generalize beyond the NL2Code domain, we evaluate PromptIntern on an additional task, which is presented in Appendix A.6.

6 Discussion

6.1 Training Overhead of PromptIntern

As PromptIntern is specifically designed to enhance the efficiency of LLMs during inference, the multi-stage design of progressive fine-tuning may introduce additional computational overhead. It is crucial to manage this overhead during the training phase to ensure that PromptIntern remains scalable and applicable in real-world, large-scale scenarios.

To illustrate the overhead of PromptIntern, we provide a detailed breakdown of the time overhead incurred during the training phase. We compare our method to the baseline *template with k-shots*, which represents common use cases. Our eval-

Table 5: Time Overhead of PromptIntern during fine-tuning.

Model	**MBPP**		**NL2F**		**NL2Bash**	
	PromptIntern	Template *with* 5-shots	PromptIntern	Template *with* 10-shots	PromptIntern	Template *with* 10-shots
GPT-4	01h 38m 54s	02h 13m 07s	03h 46m 15s	04h 32m 12s	03h 23m 18s	04h 17m 28s
GPT-3.5	01h 19m 03s	01h 48m 55s	03h 02m 29s	03h 44m 46s	02h 23m 19s	03h 09m 16s
Llama2-13B	00h 45m 27s	01h 14m 10s	01h 36m 17s	02h 16m 21s	01h 12m 48s	01h 48m 03s

uation includes the complete end-to-end process, covering dataset import and training times. All experiments were conducted on an A100x1-80G GPU. As shown in Table 5, PromptIntern consistently requires less time for both data preparation and training compared to the baseline. This efficiency is primarily due to the reduced number of input tokens per training instance during progressive fine-tuning. Furthermore, as outlined in our experimental settings, the number of training epochs for PromptIntern matches that of the direct fine-tuning baselines, ensuring no additional computational cost from extra training steps.

6.2 General model ability

Although progressive fine-tuning ensures high accuracy for domain-specific tasks, concerns may arise regarding whether this multi-step fine-tuning approach could negatively impact the overall model's generalizability. Several studies have examined fine-tuning for LLMs. (Wei et al., 2021) shows that multi-task fine-tuning enhances zero-shot and ICL capabilities. (Mosbach et al., 2023) finds that few-shot fine-tuning preserves out-of-domain generalization similar to ICL settings. However, (Wang et al., 2022) reveals that fine-tuning may overly adapt models to task-specific formats, reducing flexibility for new tasks. (Luo et al., 2023a) also explore catastrophic forgetting during continual fine-tuning.

In this work, PromptIntern is designed to enhance the efficiency of fine-tuned LLMs during inference while maintaining task-specific performance. Our approach is particularly suited for scenarios where LLMs are fine-tuned and deployed for domain-specific tasks. Although PromptIntern is currently limited to fine-tuning a single task, recent inference optimization techniques (Ye et al., 2023; Wang et al., 2023; Sheng et al., 2023) enable concurrent fine-tuning of multiple LoRA adapters while sharing a single backbone model. This allows for efficient adaptation to multiple tasks, reducing the number of parameters requiring fine-tuning. We will leave the integration of PromptIntern with these approaches in our future work.

We also provide a detailed discussion on preventing model overfitting in Appendix B.1.

7 Conclusion

In this paper, we propose PromptIntern, a prompt internalization method that internalizes repetitive prompt knowledge into LLMs parameters. We develop specific compression strategies for different components of the prompt, accompanied by a tailored progressive fine-tuning pipeline. Extensive experiments demonstrate that our method maintains comparable performance effectiveness while accelerating inference speed with less token usage.

8 Limitations

Through extensive experiments in this paper, PromptIntern has demonstrated a strong ability to reduce model costs during inference. However, as shown in Table 2, PromptIntern still exhibits a performance gap in accuracy compared to the use of original prompts (i.e., *template with k-shots*). Also, while we empirically validate the effectiveness of PromptIntern, a theoretical analysis of model parameter updates and the training pipeline complexity is still required. In addition, although the principle of PromptIntern can be generalized to most downstream NLP tasks (as demonstrated in Appendix A.6, further empirical verification is needed on more advanced tasks. In future work, we plan to conduct more evaluations on several complex tasks, including long-document summarization, question-answering in specialized technical domains, etc.

9 Ethics Statement

All datasets used in this paper are publicly available and have been reviewed to ensure they do not contain any personally identifiable information or offensive content. Additionally, experiments were conducted on computational clusters with NVIDIA A100 GPUs. It is important to note that this could have an environmental impact, and the carbon footprints were monitored in real time.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Xiang Chen, Lei Li, Ningyu Zhang, Xiaozhuan Liang, Shumin Deng, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. Decoupling knowledge from memorization: Retrieval-augmented prompt learning. In *Advances in Neural Information Processing Systems*.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model apis. *arXiv preprint arXiv:2301.08721*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith Hall, and Ming-Wei Chang. 2022. Promptagator: Few-shot dense retrieval from 8 examples. In *The Eleventh International Conference on Learning Representations*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *Advances in neural information processing systems*, 32.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Xinyi He, Jiaru Zou, Yun Lin, Mengyu Zhou, Shi Han, Zejian Yuan, and Dongmei Zhang. 2024. Conline: Complex code generation and refinement with online searching and correctness testing. *arXiv preprint arXiv:2403.13583*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Jonas Hübotter, Bhavya Sukhija, Lenart Treven, Yarden As, and Andreas Krause. 2024. Active few-shot fine-tuning. *arXiv preprint arXiv:2402.15441*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023b. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021a. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021b. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023a. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023b. Wizardcoder: Empowering code large language models with evol-instruct. *Preprint*, arXiv:2306.08568.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*.

Jesse Mu, Xiang Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.

Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-based prompt selection for code-related few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2450–2462. IEEE.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5).

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Anastasiia Razdaibiedina, Yuning Mao, Madian Khabsa, Mike Lewis, Rui Hou, Jimmy Ba, and Amjad Almahairi. 2023. Residual prompt tuning: improving prompt tuning with residual reparameterization. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6740–6757.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*.

Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2023. Tap4llm: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. *arXiv preprint arXiv:2312.09039*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Kushala VM, Harikrishna Warrier, Yogesh Gupta, et al. 2024. Fine tuning llm for enterprise: Practical guidelines and recommendations. *arXiv preprint arXiv:2404.10779*.

Yihan Wang, Si Si, Daliang Li, Michal Lukasik, Felix Yu, Cho-Jui Hsieh, Inderjit S Dhillon, and Sanjiv Kumar. 2022. Two-stage llm fine-tuning with less specialization and more generalization. *arXiv preprint arXiv:2211.00635*.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. 2023. Multilora: Democratizing lora for better multi-task learning. *arXiv preprint arXiv:2311.11501*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5621–5634.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. Recomp: Improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Zhengmao Ye, Dengchun Li, Jingqi Tian, Tingfeng Lan, Jie Zuo, Lei Duan, Hui Lu, Yexi Jiang, Jian Sha, Ke Zhang, et al. 2023. Aspen: High-throughput lora fine-tuning of large language models with a single gpu. *arXiv preprint arXiv:2312.02515*.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2022. Large language models meet nl2code: A survey. *arXiv preprint arXiv:2212.09420*.

Wei Zhao, Zhitao Hou, Siyuan Wu, Yan Gao, Haoyu Dong, Yao Wan, Hongyu Zhang, Yulei Sui, and Haidong Zhang. 2024. Nl2formula: Generating spreadsheet formulas from natural language queries. *arXiv preprint arXiv:2402.14853*.

Lecheng Zheng, Baoyu Jing, Zihao Li, Hanghang Tong, and Jingrui He. 2024. Heterogeneous contrastive learning for foundation models and beyond. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6666–6676.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

A Additional Experiments

A.1 Dataset Details

MBPP The MBPP dataset, as detailed by (Mosbach et al., 2023), consists of Python programming tasks, each accompanied by a description in natural language that has been expertly curated. The dataset is segmented into training and test sets, with 974 and 102 examples, respectively.

NL2F The NL2F dataset, as detailed by (Zhao et al., 2024), consists of 70,799 pairs of NL queries and spreadsheet formulas and covers 21,670 tables. We follow the dataset instructions (Zhao et al., 2024) to randomly split data into a training set (75%), validation set (10%), and test set (15%).

NL2Bash The NL2Bash dataset, as described by (Lin et al., 2018), comprises snippets of Bash code, each paired with a natural language description expertly curated. The dataset is divided into training and test sets, containing 8,090 and 606 examples, respectively.

A.2 Implementation Details

Fine-tuning Procedures For PromptIntern training, we adopt LoRA (Hu et al., 2021) with a rank of 32. For GPT-series and open-source model fine-tuning we train models for MBPP/NL2F/NL2Bash with 6/12/12 epochs, 16/128/128 batch size, 200/200/200 checkpoint interval, and 4096/4096/4096 context window length, respectively.

Model Inference We provide the detailed parameters we adopted during fine-tuned LLM inference: temperature equal to 0, max tokens equal to 1028, top p equal to 0.95, presence penalty equal to 0, and frequency penalty equal to 0.

Baseline Settings For prompt compression baselines comparison, we set the template compression ratio $\tau_{tmp} = 0.3$. For direct fine-tuning baselines, we apply LLMLingua-2 (Pan et al., 2024) as the default template compressor as it performs the best in Table 1.

Parameter Settings for PromptIntern
1) Number of top-k for example absorption: We set the initial k as 5/10/10 across MBPP/NL2F/NL2Bash for the initial number of few-shot examples for example absorption. During progressive fine-tuning, we decrease k linearly in the order of 5-2-0/10-5-0/10-5-0 across MBPP/NL2F/NL2Bash.
2) Number of τ_{tmp} for template compression: For the prompt compression baseline experiments, we set the final template rate to 0.3, which is used in the last stage of fine-tuning as well as inference. For the other experiments and ablation studies, we set the final template rate to 0 to achieve full internalization.

Cost Evaluation We compute the total costs based on the price shown in OpenAI Pricing[3]

Computational Resource We conduct all experiments on AzureAI Machine Learning Studio with one A100x1-80G computational cluster

A.3 Comparison with Gist Tokens

We report the comparison result of PromptIntern with Gist Tokens (Mu et al., 2024) on Table 6. Gist Tokens showcases consistent performance, with notable results in NL2Bash where it achieves a

[3]https://openai.com/api/pricing/

Table 6: Comparison with Gist Tokens (Mu et al., 2024)

Methods	MBPP			NL2F			NL2Bash		
(*Inference on Llama2-7B*)	Pass@1	Tokens	$1/\tau_{all}$	E.M.	Tokens	$1/\tau_{all}$	BLEU	Tokens	$1/\tau_{all}$
Gist Tokens	10.2	61	4.1x	17.5	342	2.4x	22.7	66	8.6x
PromptIntern	27.9	58	4.3x	40.8	339	2.4x	31.6	64	9.0x

Table 7: Speed (s/instance) Comparison of PromptIntern with Direct Fine-tuning baseline on NL2Code benchmarks.

Model	Template *with* 5-shots	Template x0.6 *with* 2-shots	Template	No template	PromptIntern
MBPP					
GPT-4	10.21	8.68	7.29	4.36	**4.17**
GPT-3.5	5.43	3.68	3.06	1.35	**1.31**
Mixtral-8x7B	4.84	3.23	3.14	1.76	**1.62**
Llama2-13B	3.17	2.54	2.19	1.08	**1.13**
Llama2-7B	2.95	2.27	1.95	0.84	**0.76**
NL2F					
GPT-4	12.47	8.43	4.16	2.12	**2.15**
GPT-3.5	8.16	5.26	2.18	1.46	**1.44**
Mixtral-8x7B	6.27	4.71	3.17	1.19	**1.20**
Llama2-13B	4.15	2.95	1.25	0.63	**0.63**
Llama2-7B	3.83	2.03	1.24	0.41	**0.39**
NL2Bash					
GPT-4	4.46	3.18	1.57	1.18	**1.18**
GPT-3.5	2.79	2.36	1.29	1.02	**1.02**
Mixtral-8x7B	2.65	2.23	1.59	1.13	**1.13**
Llama2-13B	2.21	1.96	1.41	1.05	**1.05**
Llama2-7B	1.86	1.49	1.02	0.87	**0.87**

BLEU score of 22.7, suggesting a moderate alignment with the dataset's requirements. In contrast, PromptIntern demonstrates superior performance across all metrics and datasets, particularly excelling in the NL2Bash dataset with a BLEU score of 31.6 and maintaining similar efficiency in token usage. The results demonstrate that our approach significantly outperforms the Gist token while conducting overall the same compression rate.

A.4 Effectiveness of progressive fine-tuning in PromptIntern

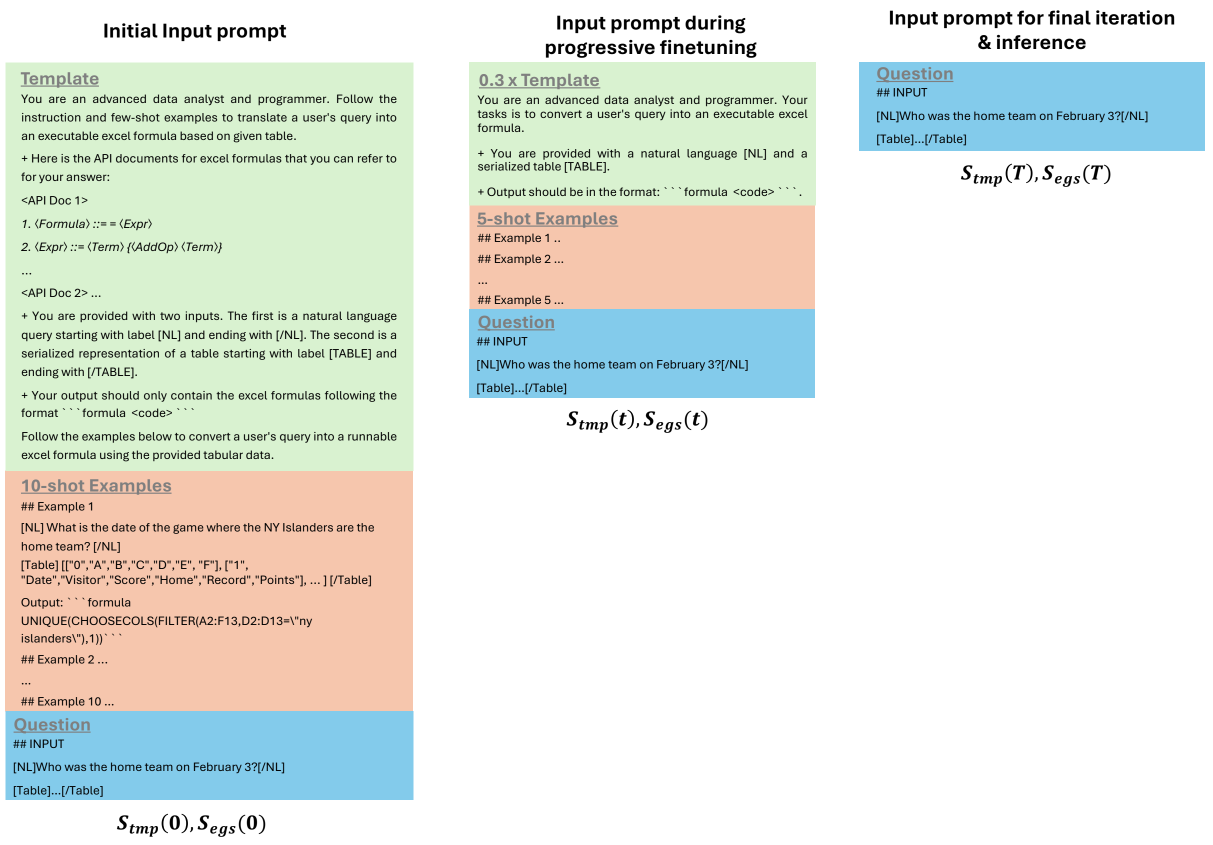
In this experiment, we compare PromptIntern with the method of direct fine-tuning with a full-loaded prompt (template plus few-shot examples) followed by inferencing with queries only (designated as *Template with k-shots** in table 8). We use the comparison to demonstrate the effectiveness of progressive fine-tuning for updating model parameters properly. The result is shown in Table 8. We can clearly observe that *Template with k-shots** has a large performance degradation compared to PromptIntern. This indicates that fine-tuned LLMs struggle to establish a proper connection between the full-length prompts used in training and the query-only prompts used during inference. It also motivates the development of the progressive fine-tuning strategy in PromptIntern. Beyond empirical experiments demonstration, we will leave the theoretical proof of the effectiveness of PromptIntern (Algorithm 1) in our future work.

Table 8: Demonstration of progressive fine-tuning in PromptIntern

Dataset (Inference on GPT3.5)	**MBPP**	**NL2F**	**NL2Bash**
Template with k-shots*	69.1	71.7	63.2
PromptIntern	**76.6**	**78.4**	**67.7**

A.5 Experiments on Inference Speed

The experimental results presented in Table 7 illustrate the low latency characteristics



```
Template
You are an advanced data analyst and programmer. Follow the instruction and few-shot examples to translate a user's query into an executable excel formula based on given table.
+ Here is the API documents for excel formulas that you can refer to for your answer:
<API Doc 1>
1. ⟨Formula⟩ ::= = ⟨Expr⟩
2. ⟨Expr⟩ ::= ⟨Term⟩ {⟨AddOp⟩ ⟨Term⟩}
...
<API Doc 2> ...
+ You are provided with two inputs. The first is a natural language query starting with label [NL] and ending with [/NL]. The second is a serialized representation of a table starting with label [TABLE] and ending with [/TABLE].
+ Your output should only contain the excel formulas following the format ```formula <code> ```
Follow the examples below to convert a user's query into a runnable excel formula using the provided tabular data.
10-shot Examples
## Example 1
[NL] What is the date of the game where the NY Islanders are the home team? [/NL]
[Table] [["0","A","B","C","D","E", "F"], ["1", "Date","Visitor","Score","Home","Record","Points"], ... ] [/Table]
Output: ```formula
UNIQUE(CHOOSECOLS(FILTER(A2:F13,D2:D13=\"ny islanders\"),1))```
## Example 2 ...
...
## Example 10 ...
Question
## INPUT
[NL]Who was the home team on February 3?[/NL]
[Table]...[/Table]
```

```
0.3 x Template
You are an advanced data analyst and programmer. Your tasks is to convert a user's query into an executable excel formula.
+ You are provided with a natural language [NL] and a serialized table [TABLE].
+ Output should be in the format: ```formula <code> ```.
5-shot Examples
## Example 1 ..
## Example 2 ...
...
## Example 5 ...
Question
## INPUT
[NL]Who was the home team on February 3?[/NL]
[Table]...[/Table]
```

```
Question
## INPUT
[NL]Who was the home team on February 3?[/NL]
[Table]...[/Table]
```

Figure 3: An Example from NL2F demonstrating how an original prompt is preprocessed through template compression and example absorption in PromptIntern for progressive fine-tuning and final inference.

of PromptIntern during inference across three datasets, MBPP, NL2F, and NL2Bash. Specifically, for the MBPP dataset, PromptIntern achieves an inference speed of 4.17 s/instance on the GPT-4 model, closely aligning with the 4.36 s/instance observed in the no template setup and far surpassing the more resource-intensive template with 5-shots configuration at 10.21 s/instance. In the NL2F dataset, PromptIntern similarly demonstrates its efficiency with an inference speed of 2.15 s/instance for GPT-4, which is nearly equivalent to the 2.12 s/instance observed without any template and significantly outperforms the elaborate template with 10-shots configuration, which achieves 12.47 s/instance. The experimental results outlined in the table also highlight the efficiency of PromptIntern in the NL2Bash dataset. Notably, for GPT-4 under the NL2Bash benchmark, PromptIntern maintains a competitive inference speed of 1.18 s/instance, matching the performance seen in the no template scenario and markedly better than the template with 5-shots setup, which records a slower speed of 4.46 s/instance. The result across three NL2Code benchmarks highlights PromptIntern 's capability to maintain competitive inference speeds while minimizing latency efficiently.

A.6 Evaluating PromptIntern on GSM8K

To demonstrate the generalization ability of PromptIntern on other domain tasks, we also test on the GSM8K dataset (Cobbe et al., 2021). We use the same experiment settings stated in our experiment setups to compare with the prompt compression baselines. The results are shown in Table 9. The result demonstrates that, under the same compression rate for inference, PromptIntern outperforms other compression baselines by 3%-34% for the arithmetic reasoning task.

Table 9: Comparison of prompt compression baselines on GSM8K.

Methods (Inference on GPT 3.5)	E.M.	Tokens
Selective Context	63.5	443
LLMLingua	83.2	452
LLMLingua-2	81.9	458
PromptIntern	**85.7**	458

B Additional Discussions

B.1 Prevention of Model Over-fitting

To prevent LLMs from over-fitting due to lengthy input prompts, such as long templates, and to mitigate over-fitting during multi-stage fine-tuning, we have implemented several strategies:

- **Prompt Selection:** For each dataset, we utilize default prompts provided by the authors or sourced from widely recognized papers. In our experiments, these prompts are applied for the baseline *template with 5/10 shots*, which outperforms other approaches and aligns with results from sources like Papers With Code. This ensures that our prompts will not cause performance degradation due to over-fitting input.
- **Model Checkpointing:** To mitigate over-fitting during fine-tuning, particularly due to excessively lengthy training epochs, we implemented a model checkpointing strategy. We save the model state every 200 training steps and evaluate each checkpoint on a separate validation dataset. This allows us to track performance changes throughout the whole training process. By comparing checkpoints, we identify the optimal iteration that achieves the best results, determining the appropriate number of training steps and epochs for our experiments.
- **Validation Monitoring:** Training is halted using an early stopping technique when the validation loss begins to rise or ceases to decline, indicating potential over-fitting. Additionally, we manually monitor the training and validation losses against the number of training steps for each experiment of PromptIntern. This visualization helps ensure that each model avoids training over-fitting by providing a clear depiction of the training dynamics and enabling timely adjustments.

C Example Demonstration

We demonstrate an example of how we schedule and pre-process an input prompt through both template compression and example absorption in Figure 3. During the initial fine-tuning phase, the input prompt will fully incorporate the template and 10-shot examples for the NL2F dataset. After a specified number of training iterations, the template will undergo compression at a rate of 0.3, and the number of examples will be reduced to five. This modified prompt is then used for the intermediate stage of fine-tuning. In the final phase, the template and few-shot examples are removed from the training prompt. It is important to note that the query remains unchanged throughout the entire progressive fine-tuning process. The prompt used in the last stage, which consists solely of the query, will also serve as the input for subsequent model inference. This method enables the fine-tuned language model to perform zero-shot inference without the need for an instruction or document template.

D Prompts

D.1 GPT-4 Generation (Baseline) instruction

For detailed prompts, please refer to Figure 4.

GPT-4 Generation Instruction

Compress the given text to short expressions, and make sure you can reconstruct it as close as possible to the original.

Unlike the usual text compression, I need you to comply with the 5 conditions below:
+ You can ONLY remove unimportant words.
+ Do not reorder the original words.
+ Do not change the original words.
+ Do not use abbreviations or emojis.
+ Do not add new words or symbols.

Compress the origin aggressively by removing words only. Compress the origin as short as you can, while retaining as much information as possible. If you understand, please compress the following text: <text to compress>
The compressed text is:

Figure 4: The instruction baseline for the baseline method GPT-4 Generation.

D.2 Prompts of PromptIntern on NL2Code

For detailed prompts of each dataset, please refer to Figures 5,6,7.

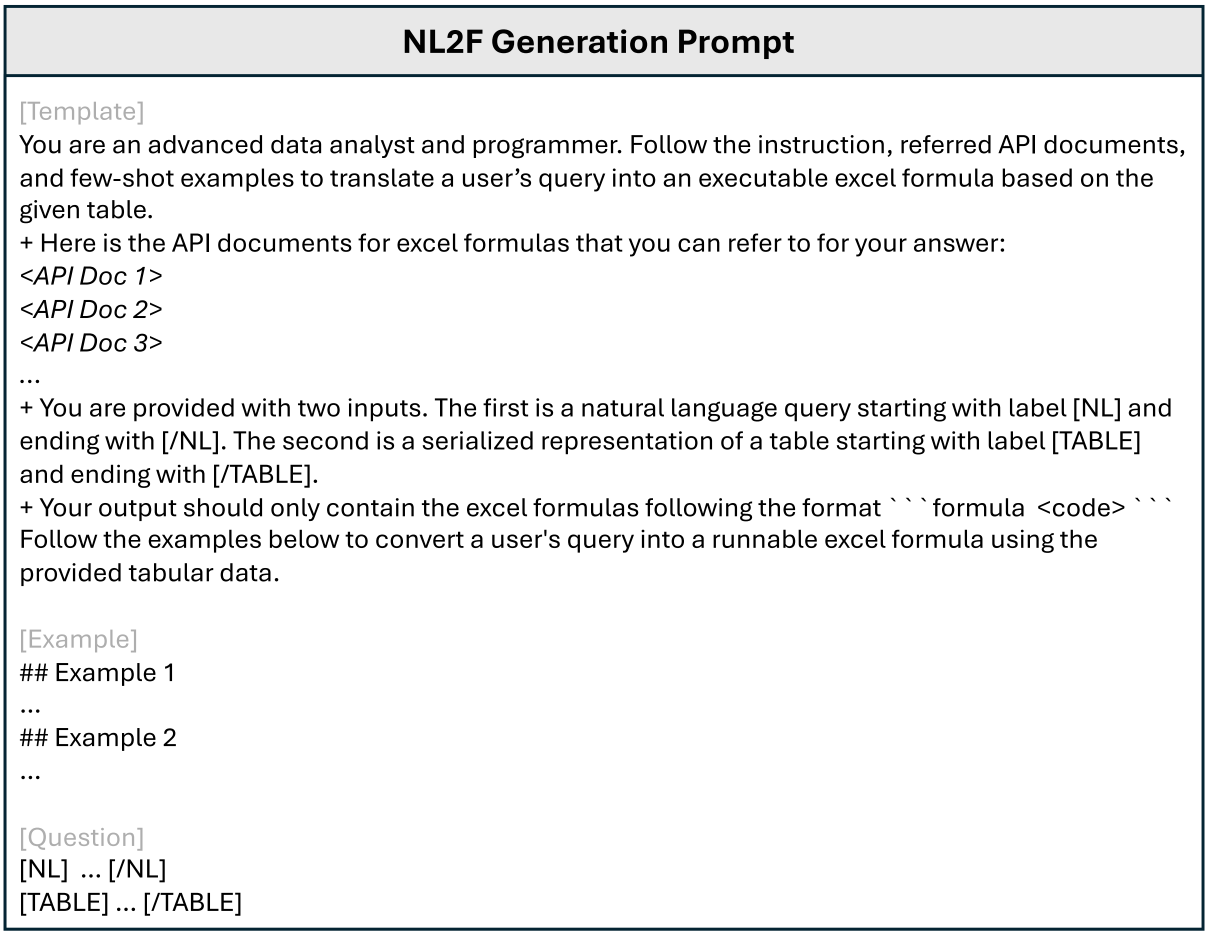
MBPP Generation Prompt

[Template]
You are an advanced Python programmer.
Read the instructions claimed below and write the corresponding Python code.
## You will be given a question describing the python function need to implement for.
## You will also be given three corresponding test cases written in Python code. They all using assert styles.
## Read the question and test cases carefully and fulfill the requirements below:
+ Your written function's name should be the same as the function name shown in the test cases.
+ Your function should take the same number of input arguments and output values as shown in the test cases.
+ Your function should handle same type of input and return the same type of value as shown in the test cases.
+ Your function should pass all the three provided test cases.
+ You can use any built-in python libraries.
+ Your output should strictly follow the format of ```python <code>```.

[Example]
## Example 1
...
## Example 2
...

[Question]
NL Question: ...
Three Test Cases: ...

Figure 5: Prompts of MBPP



NL2F Generation Prompt

[Template]
You are an advanced data analyst and programmer. Follow the instruction, referred API documents, and few-shot examples to translate a user's query into an executable excel formula based on the given table.
+ Here is the API documents for excel formulas that you can refer to for your answer:
<API Doc 1>
<API Doc 2>
<API Doc 3>
…
+ You are provided with two inputs. The first is a natural language query starting with label [NL] and ending with [/NL]. The second is a serialized representation of a table starting with label [TABLE] and ending with [/TABLE].
+ Your output should only contain the excel formulas following the format ```formula <code> ```
Follow the examples below to convert a user's query into a runnable excel formula using the provided tabular data.

[Example]
## Example 1
…
## Example 2
…

[Question]
[NL] … [/NL]
[TABLE] … [/TABLE]

Figure 6: Prompts of NL2F

NL2Bash Generation Prompt

[Template]
You are an advanced shell programmer. Follow the instruction, referred API documents, and few-shot examples to translate a user's natural language command into an executable Bash command.

+ Here is the API documents for advanced bash shell functions and commands that you can refer to for your answer:
<API Doc 1>
<API Doc 2>
<API Doc 3>
...
+ You are provided with one input. The first is a natural language query starting with label [NL] and ending with [/NL].
+ Your output should only contain the excel formulas following the format ```bash <code> ```

Follow the examples below to convert a user's query into a runnable bash command.

[Example]
## Example 1
...
## Example 2
...

[Question]
[NL] ... [/NL]

Figure 7: Prompts of NL2Bash